

Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control

Jeff Burke

Paolo Gasti

Naveen Nathan

Gene Tsudik

ABSTRACT

Instrumented environments, such as modern building automation systems (BAS), are becoming commonplace and are increasingly interconnected with (and sometimes by) enterprise networks and the Internet. Regardless of the underlying communication platform, secure control of devices in such environments is a challenging task. The current trend is to move from proprietary communication media and protocols to IP over Ethernet. While the move to IP represents progress, new and different Internet architectures might be better-suited for instrumented environments.

In this paper, we consider security of instrumented environments in the context of Content-Centric Networking (CCN). In particular, we focus on building automation over Named-Data Networking (NDN), a prominent instance of CCN. After identifying security requirements in a specific BAS sub-domain (lighting control), we construct a concrete NDN-based security architecture, analyze its properties and report on preliminary implementation and experimental results. We believe that this work represents a useful exercise in assessing the utility of NDN in securing a communication paradigm well outside of its claimed *forte* of content distribution. At the same time, we provide a viable (secure and efficient) communication platform for a class of instrumented environments exemplified by lighting control.

1 Introduction

The Internet has clearly proven to be a tremendous global success. Billions of people worldwide use it to perform a wide range of everyday tasks. It hosts a large number of information-intensive services, involves enormous amounts of content created and consumed over the Web, and interconnects untold millions of wired, wireless, fixed and mobile computing devices.

Since Internet's inception, the amount of data exchanged over it has witnessed exponential growth. Recently, this growth intensified due to increases in: (1) distribution of multimedia content, (2) popularity of social networks and (3) amount of user-generated content. Unfortunately, the same usage model that fueled Internet's success is also exposing its limitations. Core ideas of today's Internet were developed in the 1970-s, when telephony – i.e., a point-to-point conversation between two entities – was the only successful example of effective global-scale communication technology. Moreover, original Internet applications were few and modest in nature, e.g., store-and-forward email and remote computer access.

The world has changed dramatically since the 1970-s and the Internet now has to accommodate new services and applications as well as different usage models. To keep pace with changes and move the Internet into the future, several research efforts to design new Internet architectures have been initiated in recent years.

Named-Data Networking (NDN) [30] is an on-going research

project that aims to develop a candidate next-generation Internet architecture. NDN exemplifies the so-called Content-Centric approach [18, 24, 28] to networking. It explicitly names content instead of physical locations (i.e., hosts or network interfaces) and thus transforms content into a first-class entity. NDN also stipulates that each piece of named content must be digitally signed by its producer. This allows decoupling of trust in content from trust in the entity that might store and/or disseminate that content. These NDN features facilitate automatic caching of content to optimize bandwidth use and enable effective simultaneous utilization of multiple network interfaces.

NDN's long-term goal is to replace TCP/IP. In order to succeed, NDN must prove that it can be used to efficiently implement all kinds of communication commonly performed over IP today and envisaged for the near future. NDN has been shown as a viable architecture for content distribution [24] as well as real-time [23] and anonymous communication [6]. However, it remains unclear how NDN would fare in the context of other, less content-centric, communication paradigms, such as: cyber-physical systems (CPS), group communication (e.g., conferencing) and instrumented environments (e.g., building automation).

1.1 Building Automation and Lighting Control

Building Automation Systems (BAS) provide a hardware and software platform for control, monitoring and management of: heating, ventilation and air conditioning (HVAC), lighting, water, physical access control and other building components. BAS are subject to several important current trends with security implications:

- Increasing use of IP and Ethernet for industrial control, often over commodity wiring and network hardware.
- Convergence of previously separate networks for automation and IT enabled by this new common infrastructure.
- Increasing interest in cyber-physical systems (CPS) that leverage internetworking of physical and digital elements to enable and develop new types of applications.

In general, BAS offers an interesting and challenging application domain for NDN because content-centric networking is generally discussed in terms of its improvements to content retrieval, as opposed to control, actuation, or remote execution. In the domain of BAS, we focus on lighting control as the initial test platform for the design and implementation of control communication over NDN. This choice is based on three reasons: (1) lighting represents a broad class of actuators while incurring limited physical safety concerns; (2) prevalence of IP-based control of lighting fixtures in new architectural and entertainment deployments; (3) access to comparisons with IP- and serially-controlled systems.

In designing an NDN lighting control framework, our goals are:

1. Satisfy low latency requirements for communication between software (or hardware) controllers and lighting fixtures.
2. Use NDN content naming to address all components of the system, with names related to their identity or function rather than a combination of addressing that spans layers and systems (e.g., VLAN tag, IP address of gateway, port of protocol, address of fixture) as in current implementations.
3. Given that NDN makes widespread use of content signatures, identify every entity in the system by a distinct public key.
4. Control access to fixtures via authorization policies, coupled with strong authentication. (Current BAS and lighting systems typically rely on physical or VLAN-based segregation for security, making interoperability with IT systems challenging, configurations brittle to change, and requiring advanced networking expertise to set up and maintain.)
5. Use NDN naming itself to reflect access restrictions, rather than require a separate policy language. The main motivation is that a namespace is consistently accessible within any NDN-compliant device or process. This obviates the need for application-specific access control protocols.
6. Develop security mechanisms suitable for low-power systems, initially targeting cell-phone class devices with a planned transition to microcontrollers typical of IP-connected lights today, such as the Phillips Color Kinetics ColorBlaze, that uses a 72-MHz ARM processor.

1.2 Securing Lighting Control

Current lighting control installations, especially in non-critical facilities, tend to rely on network segregation and/or VLANs and VPNs to isolate their control traffic from general-purpose IP communication. Several trends, such as increasing emphasis on energy management and the “smart grid” suggest that this will not remain a viable approach in the future: Instrumented environments have increasing reliance on the Internet for patches and updates, remote access, data gathering, and application integration, as well as increasing opportunities for integration in homes and other environments without enterprise-level security. Consequently, we believe that, in the near future, it will become increasingly difficult to provide effective security by network separation alone: the most compelling applications depend on interfaces across system and network boundaries. VLANs, IP subnetting, and other network configurations spread addressing information across network layers in a way that is rarely meaningful to end-users or application developers.

Besides lower complexity and greater interoperability, running lighting control applications over public networks brings certain advantages. First, there is no need to design, deploy and manage a separate network infrastructure, since lighting control can benefit from high-speed, low-latency, fault-tolerant networks already deployed for general-purpose communication needs. Second, lighting control can be physically distributed, with devices spanning buildings and sites, and applications accessing them from a variety of locations. Third, separate (often esoteric and proprietary) security measures common in today’s lighting control would be unnecessary, due to availability of standardized security features and techniques.

1.3 Focus

This paper is focused on securing lighting control systems running over NDN. As mentioned above, allowing control messages to reach actual lighting fixtures (as opposed to dedicated controllers) imposes strict performance constraints, in addition to more general requirements of availability and fault tolerance. While general BAS might tolerate variable delays up to a few seconds for actuating or sensing slowly changing systems, latency requirements for lighting control are stricter and represent an overlap with industrial and

process control. To provide a sense of “real-time” interaction, architectural lighting might require execution of commands within a few hundred milliseconds of pressing a switch, or updates close to 44Hz DMX refresh rate [9] to achieve a smooth fade from one value to another. By designing and implementing a secure lighting framework suitable for such low latency systems coupled with a meaningful namespace, we target a hybrid design space that corresponds to the so-called “thin waist” for highly heterogeneous BAS of the future.

Organization: We proceed with NDN overview in Section 2. It is followed by the description of a base-line lighting control protocol in Section 3. The same section introduces our framework. Implementation details and performance evaluation results are discussed in Section 4. Section 5 contains the security analysis. Next, Section 6 summarizes related work. The paper concludes with future work agenda and a summary in Section 7.

2 Overview of NDN

NDN [30] is a communication architecture based on named content. Rather than addressing content by its location, NDN refers to it by name. A content name is composed of one or more variable-length components. Component boundaries are explicitly delimited by “/”. For example, the name of a CNN news content might be: `/ndn/cnn/news/2011aug20`. Large pieces of content can be split into fragments with predictable names: fragment 137 of a YouTube video could be named: `/ndn/youtube/video-749.avi/137`.

Since the main abstraction is content, there is no explicit notion of “hosts” in NDN. (However, their existence is assumed.) Communication adheres to the *pull* model: content is delivered to consumers only upon explicit request. A consumer requests content by sending an *interest* packet. If an entity (a router or a host) can “satisfy” a given interest, it returns the corresponding *content object*. Interest and content are the only types of packets in NDN. A content packet with name *X* in NDN is never forwarded or routed unless it is preceded by an interest for name *X*.¹

When a router receives an interest for name *X* and there are no pending interests for the same name in its PIT (Pending Interests Table), it forwards the interest to the next hop, according to its routing table. For each forwarded interest, a router stores some state information, including the name in the interest and the interface on which it was received. However, if an interest for *X* arrives while there is already an entry for the same name in the PIT, the router collapses the present interest (and any subsequent ones for *X*) storing only the interface on which it was received. When content is returned, the router forwards it out on all interfaces from which an interest for *X* has been received and flushes the corresponding PIT entry. Note that, since no additional information is needed to deliver content, an interest does not carry a “source” address. Any NDN router can provide content caching; its magnitude is limited only by resource availability. Consequently, content might be fetched from routers caches, rather than from its original producer. (Hence, no “destination” addresses are used in NDN). Further details of NDN architecture can be found in [24].

NDN deals with content authenticity and integrity by making digital signatures mandatory for all content. A signature binds content with its name, and provides origin authentication no matter how, when or from where it is retrieved. Public keys are treated as regular content: since all content is signed, each public key content is effectively a “certificate”. NDN does not mandate any particular certification infrastructure, relegating trust management

¹Strictly speaking, content named $X' \neq X$ can be delivered in response to an interest for *X*, but only if *X* is a prefix of *X'*.

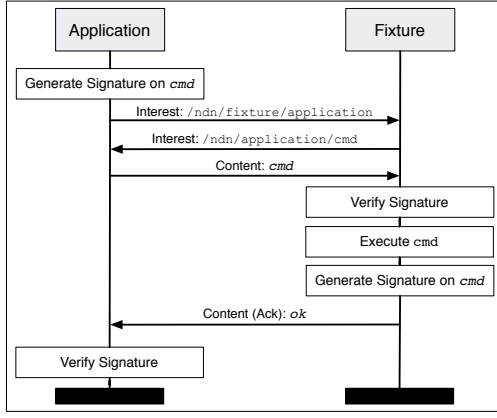


Figure 1: Base-line protocol.

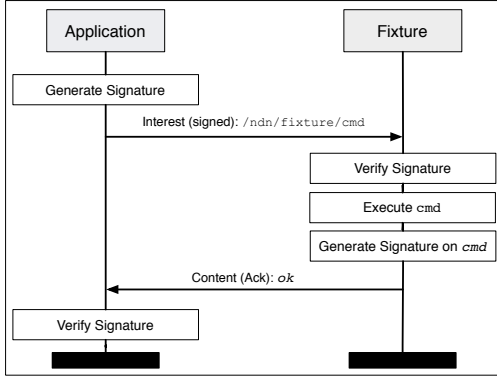


Figure 2: Protocol with authenticated interests.

to individual applications. Private or restricted content in NDN is protected via encryption by the content publisher.

3 Lighting Control over NDN

Our setup involves four parties: a configuration manager (CM), one or more fixtures (Fix), one or more applications (App) and an authorization manager (AM). CM is in charge of the initial fixture configuration. This includes, on a per-fixture basis: assigning a fixture its NDN namespace, installing a trusted public key (owned by AM) that identifies the local domain, and giving a fixture its identity represented by a unique public key. Note that, in NDN parlance, “namespace” refers to content published by some entity, whereas, “identity” refers to a public key associated with some entity that publishes content. AM determines which applications are allowed to access each fixture, signs applications’ public keys and (optionally) issues signed access control lists. While CM and AM represent distinct functions, in practice, they are likely to be physically co-located.

3.1 Base-Line Protocol

We start by observing that NDN can be easily used to securely implement basic lighting control without requiring any new features or components. As shown in Figure 1, when application App needs to send a command to fixture Fix, the base-line protocol works as follows:

1. App creates (and signs) a new content object *cmd* containing the desired command.

2. App issues an interest *int_A* with a name in Fix’s namespace that references the name of *cmd*.
3. Fix receives *int_A*, stores it in its PIT, and issues an interest *int_F* for the name of *cmd*.
4. App receives *int_F* and responds with *cmd*.
5. Fix receives *cmd*, (1) checks its access control list to determine if App is authorized to execute the command in *cmd*, (2) verifies the signature of *cmd*, (3) executes the command, and (4) replies with an acknowledgement (from here on abbreviated as “ack”) in the form of a new signed content object *ack*. Finally, Fix flushes *int_A* from its PIT.
6. App receives *ack* and verifies its signature.

The main drawback of this protocol is its high latency and bandwidth overhead: a single command requires 4 rounds and 4 messages, instead of the ideal 2 rounds/messages (Also, as is well-known, protocol robustness suffers and complexity increases with the number of rounds.) Thus, this approach is a poor match for delay-sensitive lighting control.

Alternatively, Fix could continuously issue interests that solicit App’s commands. This way, whenever App issues a new command, it does so by simply satisfying the most recent interest. While this approach would address the latency issue of the base-line protocol, it introduces new problems.

First, Fix would have to always issue one interest per each App allowed to control it. In an installation with multiple applications (*m*) interacting with a large number of fixtures (*n*), the overhead of periodic $O(mn)$ interests would be significant. Also, an application would be unable to generate a rapid burst of commands to the same fixture, since the latter would operate in a lock-step fashion. (In other words, App can only issue a new command after it receives an interest from Fix).

3.2 Whither Authenticated Interests?

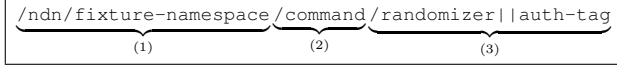
We now consider another approach that, at least in principle, violates the tenets of NDN. Recall that NDN stipulates that all content objects must be signed. Each entity implementing the NDN protocol stack must be able to verify content signatures. Interests, however, are not subject to the same requirement. One reason for this design choice is efficiency: public-key signature generation and verification is expensive. Moreover, signatures from different parties prevent straightforward interest aggregation. Another reason is privacy: traditional public-key signatures carry information about the signer. There are, however, applications that could benefit from authenticated interests and control of building systems like lighting seems to be one.

Authenticated interests can be implemented using both public key and symmetric authentication mechanisms, i.e., signatures and message authentication codes (MACs), respectively. For the sake of generality, we refer to the output of both as *authentication tags*. Due to the flexibility of NDN naming, where name components can be application-determined and are opaque to the network, an authentication tag can be placed into an NDN name as a *bona fide* component thereof. This way, an authentication tag becomes transparent to NDN routers and only the target of the command would interpret and verify it.

While MACs obviously perform much better than signatures, they make auditing difficult: a MAC cannot be attributed to one party. For this reason, if there is an auditing requirement – and if timing constraints allow – we prefer signature when fixtures are used as actuators (i.e., when commands generate feedback). Whereas, MACs are naturally preferred when fixtures act as sensors, i.e., when applications retrieve information from them. Regardless of their type, computation of authentication tags must be random-

ized to ensure uniqueness, based on either nonces or timestamps. However, this would inhibit aggregation of authenticated interests, since each authentication tag would be distinct; hence, no two names would be the same, with overwhelming probability.

In general, using authenticated interests is fairly straightforward, as illustrated in Figure 2. CM configures each fixture with a specific namespace and AM assigns a set of rights to each application, tied with the application public key or to a symmetric key shared with the fixture. The name reflected in an authenticated interest would contain three parts: (1) prefix part (used for routing) that corresponds to the fixture namespace, (2) actual command, and (3) randomizer (nonce or timestamp) along with the authentication tag computed over the rest of the name:



The idea is that, when Fix receives such an interest, it verifies the authentication tag, executes the command and replies with a signed ack as content. The first task (verifying the authentication tag) is simple only if one application controls the fixture or if part 1 of the name somehow uniquely identifies the requesting application. Whereas, if multiple applications are allowed to issue the same class of commands to a given fixture and use the same type of authentication tags, the fixture would need to determine the exact application by repeatedly verifying the authentication tag. This could translate into costly delays. This issue can be easily remedied by overloading NDN names even further and including another explicit part that identifies the requesting application (or its key). Another drawback of this (and the base-line) approach is that the fixture needs to sign, in real-time, the acknowledgement, which is represented as content. Since a typical fixture is a relatively anemic computing device, signature generation might involve non-negligible delays.

On the other hand, authenticated interests offer much faster (2-message/2-round) operation than the base-line protocol described in the previous section. Furthermore, an application can issue multiple commands to the same fixture in rapid succession, i.e., without waiting for an ack. (However, note that this could have negative consequences if closely-spaced interests arrive out of order).

3.3 Secure Lighting Control Framework

Based on the preceding discussion, we conclude that a more specialized approach to secure lighting control over NDN is necessary in order to obtain reasonable performance while adhering to NDN principles. To this end, we construct a security framework that includes:

- A trust model wherein public keys are associated with NDN namespaces. The framework relies on this functionality to determine the entity that “owns” a particular namespace. For example, this allows us to ensure that a content object issued by a fixture in response to a command has been generated by the correct party.
- A syntax for key attributes and access control policies that binds a public key with its attributes, as determined by the signer (certifier) of this key.
- A protocol that defines how fixtures are initialized and how applications and fixtures handle authenticated commands.

In the design of our framework, we consider an adversary that can control the communication channel between App and Fix, i.e., it can record, drop, modify, inject, delay, or replay any packet. The goal of the adversary is to (1) produce a command of its choice that is successfully executed by Fix; or (2) undetectably delay, or replay

legitimate commands from App; or (3) provide an acknowledgment to App for a command that has not been executed.

3.4 Trust Model

NDN does not mandate the use of any particular trust model: each application is free to adopt the trust model that best suits it. Our trust model allows an entity (e.g., applications and fixtures) to publish content only in its namespace or any of its children. ($name_A$ is a child of $name_B$ if the latter is a prefix of the former).

Our trust model implements this restriction using public-key cryptography. Zero or more public keys are associated with each namespace. A content object published under namespace $name$ must be signed using the key associated to $name$ or any of its ancestors.

A trusted third party (TTP) – e.g., AM – generates the key-pair $K_{root} = (pk_{root}, sk_{root})$ and distributes pk_{root} . This public key is used as root of trust; a signature on a content object computed using sk_{root} is always accepted. In order to associate pk_P , belonging to producer P , with namespace $name_P$, TTP publishes, under $name_P/key$, a content object containing pk_P .

P can delegate a key $sk'_P \neq sk_P$ to sign content in namespace $name_P/sub\text{-}namespace$ by publishing the corresponding public key pk'_P under $name_P/sub\text{-}namespace/key$. This mechanism allows TTP to delegate some of its certification capabilities to each producer.

P can prove to anyone its ownership of a key linked to $name_P$ through a simple challenge-response protocol. The challenger issues an interest for a content object with name $name_P/nonce$ where $nonce$ is a fresh random string selected by the challenger. P is able to respond with a valid content object only if it owns the signing key linked to $name_P$, one of its ancestors, or TTP’s signing key.

While our implementation is based on RSA signatures, hierarchical identity-based signature (HIBS) [14] schemes represent a viable alternative. A HIBS scheme is a signature scheme where any string can be a public (i.e., verification) key. Private (signing) keys are generated by a key generation center. Given a signing key sk corresponding to a string s , sk is also a valid signing key for any string $s||t$ where “||” denotes string concatenation. Moreover, sk can be used to compute a new signing key sk' corresponding to $s||u$ for any string u .

With HIBS, the public key corresponding to a namespace $name$ is the string representing $name$. TTP acts as a key generation center and issues signing keys to producers, each key corresponding to a namespace.

The main drawback of an implementation based on HIBS is the lack of support from the current NDN codebase. As such, routers cannot verify HIBS signatures on content objects. For this reason, our lighting control prototype relies on RSA signatures.

3.5 Key Attributes & Access Control Policies

All attributes of a public key are expressed using the name under which such key is published. Each attribute is a name/value pair expressed as two consecutive namespaces: the first indicates a key attribute name, and the second – its value.

Recall that an NDN content object is bound to its name by a public-key signature. According to our trust model, such a signature must be issued either by the TTP or by the owner of the namespace that contains the public key. The set of attributes defined in our framework is detailed in table 1. Applications can extend this set with new attributes. For example, a public key pk_P published under `/ndn/uci/ics/432B/domain/lighting-domain-1/appname/light-board-1/access/full-access/expires/20151231235959Z/key` specifies that pk_P belongs to application *light-board-1* in domain *lighting-domain-1*, that has “full

Attribute	Description
domain	application's domain
apptime	application identifier
access	application's permissions on the fixture
expires	expiration date in generalized time notation: (YYYYMMDDHHMMSSZ)

Table 1: Attributes

access” to fixtures in such domain. pk_P – when published under such name – is considered invalid after December 31, 2015.

An attribute name can appear more than once with different values. The combined attribute value is computed as the intersection of all the instances of such attribute. As with any NDN data packet, the issuer of content object with payload pk_P is specified in the content object's key *locator* field.

3.6 The Protocol

We now introduce the protocol for controlling NDN-connected light fixtures. The protocol is composed of three sub-protocols: bootstrapping, application authorization and control.

Bootstrap. New fixtures must be *paired* with CM and *bootstrapped* in order to be able to receive commands from applications. The pairing process consists of the distribution of a (short) symmetric key from Fix to CM. For example, in our implementation CM scans a barcode on Fix's enclosure, that represents a symmetric key factory-installed on Fix. Next, CM initializes Fix.

Fixture initialization consists of selecting an NDN name for Fix, (loosely) synchronizing CM and Fix clocks and installing (on Fix) a trusted public key that belongs to AM. This public key identifies the domain under which Fix operates. CM then communicates a signing key-pair to Fix.² This key-pair is linked to Fix's namespace, and it represents the identity of Fix. Additionally, CM can specify the NDN name of one or more ACLs that Fix must use to determine applications' permissions. At this time, Fix also generates a long-term secret master key k_{Fix} . This key is optionally used later to derive application-specific symmetric keys (i.e., k_{App}) for authentication purposes. Once Fix is correctly initialized, it responds with the current time and a hash of all the information exchanged during bootstrap.

Application Authorization. AM grants control privileges to an application by signing the latter's public key. Given pk_{App} belonging to App and intended permissions $perm_{App}$, AM first constructs a namespace $name_{App}$ containing “access/ $perm_{App}$ ”, as specified in Section 3.5. Then it signs pk_{App} and publishes it (as content) under $name_{App}$. Any fixture under control of AM can verify that App owns permission $perm_{App}$ by asking it to prove ownership of namespace $name_{App}$, as in Section 3.4.

Control Protocol. The protocol is designed for resource-constrained fixtures interacting with a large number of applications. Thus, we aim to minimize computation and communication costs and amount of memory required to perform interest authentication. We avoid storing per-application long-term information (e.g. application keys) on each fixture. A fixture stores a constant amount of state for each application currently interacting with it. We emphasize that, in order to issue and verify commands, applications and fixtures do not need to communicate with either CM or AM.

Application App, that owns a key distributed under $name_{App}/key$, issues an interest with command cmd for fixture Fix with NDN name $name_{Fix}$ as follows:

² Alternatively, Fix can generate a signing key-pair and communicate the resulting public key to CM.

$name_{Fix}/name_{App}/cmd/auth-token$

The string “ cmd ” represents a fixture-specific command. Since our framework does not specify any particular format for commands, this string is simply treated as an *opaque* binary field. For example, a simple command could be: “on” or “off”, while a more complex one could be: “intensity/+10/rgb-8bit-color/F0FF39”.

The field *auth-token* encodes the command authentication token, constructed as: $state || authenticator$. The first part represents state information required to prevent timing and replay attacks. It is, in turn, composed of: sequence number, timestamp and estimated round-trip time (RTT) between App and Fix. The *authenticator* part is a signature or a MAC. In either case, it is computed over: “ $name_{Fix}/name_{App}/cmd/state$ ”. App signs its commands using the private counterpart of $name_{App}/key$. The key used to compute and verify commands authenticated with MAC is negotiated between App and Fix as detailed below.

When a fixture receives an interest $name_{Fix}/name_{App}/cmd/auth-token$, it determines whether to execute cmd , as follows:

1. Verifies that cmd is well formed.
2. Examines attributes in $name_{App}$ to determine whether App is allowed to issue cmd (e.g., Fix check whether $name_{App}$ expiration date and access fields).
3. If available, uses a local or remote ACL specified by CM during the bootstrap phase.
4. Verifies the state of the command. First determines whether the interest is current (also using the estimated RTT value as additional information). Then, if it has no record of previous commands from App, Fix extracts the sequence number from *auth-token* and stores it as: ($name_{App}$, *sequence number*). Otherwise, it checks that the stored sequence number is lower than the one in *auth-token*.
5. Verifies *authenticator* – signature or MAC on the interest. In case of signature, Fix retrieves public key $name_{App}/key$ and stores it in its local cache.

If a pair ($name_{App}$, *sequence number*) stored by Fix is not updated for a predetermined amount of time, it is considered stale and deleted. This way, at any given time, a fixture only retains state information related to active applications.

$E_k(\cdot)$	Symmetric encryption algorithm (e.g. AES)
$H(\cdot)$	A collision-resistant hash function (e.g. SHA-256)
$MAC_k(\cdot)$	Message authentication code (e.g. HMAC-SHA-256)
$PRF_k(\cdot)$	Pseudorandom function
$name_i$	NDN name associated with entity i

Table 2: Notation

Symmetric Authentication. By default, fixtures and applications authenticate commands and acks using public-key signatures. However, for performance reasons, they can switch to MAC-s at anytime, which requires establishing a shared secret key. Recall that, at bootstrap, Fix generates a long-term secret key k_{Fix} . When App asks Fix to switch to symmetric authentication, the latter uses k_{Fix} to compute an application-specific key k_{App} . After verifying that App owns the namespace $name_{App}$ (see Section 3.4), Fix computes $k_{App} = PRF_{k_{Fix}}(name_{App})$. Then, Fix sends k_{App} to App encrypted under public key $name_{App}/key$. Note that Fix does not need to store these application-specific symmetric keys: it can compute k_{App} from k_{Fix} whenever needed. Therefore, the amount of symmetric-key-related state stored by Fix does not depend on the number of authorized applications.

3.7 Command and Ack Privacy

We now consider privacy of commands and acks. An eavesdropper may want to learn various parameters in a command or a corresponding ack. An application conceals this information by encrypting the command before constructing the name that goes into an interest. If symmetric authentication is used, the command encryption key is derived from the MAC key. If the interest is signed, the command is encrypted using Fix's public-key. NDN already provides a framework for content encryption [36] that we use to implement ack privacy.

For efficiency reasons, we do not conceal sizes of either commands or acks, thus potentially allowing the adversary to distinguish among types of commands. Although it is easy enough to introduce padding (though incurring costs), more sophisticated attacks exploiting *side channels* (e.g., time required for a fixture to respond to a command or some other observable feedback) are much more difficult to address. Since one the main goals of our approach is generality, we do not implement any countermeasure for this class of attacks.

3.8 Ack Authentication

While not typical today for lighting, we desire that a fixture and other actuators should provide feedback after processing a command. In our NDN context, this naturally results in a closed-loop control system and allows NDN routers to flush PIT entries corresponding to processed commands (interests). For obvious security reasons, acks must be authenticated. (NDN anyhow requires all content to be signed). However, in resource-constrained environment of light fixtures, the cost of computing per command (or per ack) public-key signatures is quite high, especially, considering that a fixture might receive numerous closely-spaced commands. For this reason, we propose an NDN extension allowing fixtures to efficiently produce authenticated command acks.

A natural and efficient alternative to public-key signatures are symmetric MACs. An application and a fixture could share a key, and use to authenticate acks, i.e., replace a signature on the content object (that carries the ack) with a MAC. Unfortunately, this approach is unworkable if fixtures and applications communicate through a public network, specifically, if any NDN routers are involved in Fix-App communication. Since MACs are not publicly verifiable, an intervening NDN router has no means of authenticating MAC-d content and may simply drop it.

Nonetheless, we view NDN as work-in-progress. Thus, we consider end-to-end MAC-based symmetric authentication of content as an alternative to publicly verifiable signatures and include it in the implementation.

Next, we describe two techniques that allow public verifiability of acks without requiring public-key operations by fixtures, applications or NDN routers.

Encryption-based Authentication. This technique assumes that App and Fix share a symmetric key k , itself derived from Fix-App shared key k_{App} , which is generated at bootstrap time. To begin, App generates a random s -bit value x and, using a block cipher E with block size s , computes $y = E_k(x)$, $z = H(x)$ where E is used in the ECB mode. App includes the pair (z, y) as part of the command to Fix. Recall that this command is represented as an NDN interest and, on the path to Fix, it leaves state in all intervening NDN routers.

Having received an interest, Fix extracts x' from y as $x' = E_k^{-1}(y)$ and re-computes $z' = H(x')$. If $z' \neq z$, then Fix aborts; otherwise, it issues an ack in the form of an empty content object with x as a signature.

Although x is clearly not an actual signature, this technique allows public verifiability. An NDN router that observes the (ack) content object carrying x must have a corresponding interest (and therefore z) in its PIT. It can efficiently determine the relationship between the interest and the content by checking whether $H(x) \stackrel{?}{=} z$.

Commands that are not acknowledged can be retransmitted until they time out. Once a command expires, it must be reissued using a new challenge. Despite public verifiability, App cannot prove to a third party that it successfully interacted with Fix. This is because App can unilaterally produce any number of challenge/response pairs without any interaction with Fix. This motivates a stronger (hash-chain-based) technique described below.

Hash-Chain-based Authentication. The present technique allows App to prove to any third party that it successfully interacted with Fix, with no need for any shared keys; in particular, Fix-App interactions become auditable. This is very useful in certain circumstances. For example, consider an emergency lighting system: an emergency response team can issue an "alarm" command to each fixture and turn on all lights. In case of post-incident investigation, the emergency response team can prove that it issued required commands by producing acks from the appropriate fixtures. Another example is a building security system, that, in case of an alarm, needs to trigger security lights; proving that lights were turned on correctly can be crucial for subsequent evidence gathering.

However, due to its lock-step feature (see below), this technique is designed for infrequent use. At the same time, it is appropriate whenever traditional signature computation by fixtures is too costly – i.e., when the framework is instantiated on resource-constrained devices that cannot perform public-key operations.

Hash-chain signature schemes are particularly appealing for low-powered devices due to the reduced resource burden of traditional signature schemes, which have lead to several protocols incorporating their use [3, 8, 5, 38]. Our hash-chain-based authentication method is somewhat similar to Server-Supported Signatures (S^3) scheme [3]. However, unlike S^3 , it does not rely on the server and does not involve the use of public key cryptography in the generation or verification of ack-s. However, it also does not provide all functionality offered by S^3 ; in particular, our method can not be used to authenticate an arbitrary payload.

We denote recursive application of hash function $H(\cdot)$ i times to input x as $H^i(x)$, i.e., $H^0(x) = x$ and $H^i(x) = H^{i-1}(H(x))$. A *hash chain* is a sequence $\{x, H^1(x), \dots, H^\ell(x)\}$ for some secret x and $\ell > 0$. The last link in the chain $H^\ell(x)$ is called an *anchor*. Hash chains are commonly used for authentication as follows: Alice selects a random secret value x and sends $H^\ell(x)$ to Bob through an authenticated channel. Bob authenticates Alice by challenging her with the value he currently stores – $H^i(x)$ for some $i \leq \ell$. Alice responds with a pre-image $H^{i-1}(x)$ of $H^i(x)$. Since $H(\cdot)$ is assumed to be collision-resistant, knowledge of $H^j(x)$ with $j \leq i$ is required to respond to the challenge.

In our protocol, Fix generates a secret seed x for a hash chain C of length ℓ . The anchor $H^\ell(x)$, along with other parameters (including ℓ) is signed by AM or by Fix itself, with its own private key. The result is essentially a certificate valid for up to ℓ signatures. Fix sends this certificate to App, either off-line or on-line, whenever it depletes the previous hash chain. For each command, App includes the last value $H^i(x)$ received from Fix. Fix responds to a command with an ack containing: σ , $H^\ell(x)$ and $H^{i-1}(x)$. App can then prove to a third party that it successfully interacted with Fix at least $\ell - i$ times by revealing anchor $H^\ell(x)$, the certificate (for the anchor) and the last preimage $H^i(x)$ received from Fix.

The main drawback is that App cannot issue a new command until it receives an ack from Fix for its last command. This lock-

Operation	Intel Core2Duo	ARM Cortex A8
Create auth. command (RSA-1024)	1.981 ms	21.553 ms
Verify command (RSA-1024)	0.096 ms	0.435 ms
Compute HMAC key from Fix’s secret	0.005 ms	0.046 ms
Create auth. command (HMAC)	0.006 ms	0.067 ms
Verify command (HMAC)	0.013 ms	0.152 ms

Table 3: Performance of RSA and HMAC authenticated commands on Intel and ARM platforms.

step approach makes the technique unsuitable if App needs to have multiple commands *in flight* for any given fixture. Moreover, Fix must use a different hash chain for every controlling application.

Packet Loss. Either the interest from App to Fix or the corresponding ack might be lost. Clearly, App cannot distinguish between the two cases. After issuing a command, if the ack is not received, App continues to issue *the same* interest for a predetermined amount of time. If still no ack is received, the command is aborted and App and Fix fall back to authenticating acks through regular public-key signatures. Fix can easily determine if a received interest corresponds to a retransmission or to a new command by checking whether the preimage of the challenge $H^i(x)$ in the interest has already been revealed. If the command is a re-transmission and the original has already been acknowledged, Fix simply issues a new ack containing the pre-image of $H^i(x)$; this requires no computational effort.

4 Prototype Evaluation

In order to evaluate the performance of the proposed architecture, we implemented a library – called *NameCrypt* – designed for lighting control in a theatrical environment. We also deployed it in an actual theatrical lighting installation. In this setting, applications and lighting fixtures interact over a local-area network.

Our setup involves three applications: (1) a sequencer that outputs pre-generated patterns, (2) a controller that uses algorithmic patterns and (3) a fader. These applications control eleven lights, connected to five embedded devices (described below). The embedded devices, in turn, are connected to the lights using KiNet, a proprietary Philips protocol which runs on TCP/IP over ethernet.

4.1 Implementation

NameCrypt provides low-level functionality required by our protocol on top of CCNx. It is implemented in C interfacing to OpenSSL for cryptographic services and to CCNx for transport. The target platform of the lighting fixture is an off-the-shelf embedded device based on the Gumstix Overo Air [19] computer-on-a-module. This device is running a 600 MHz Texas Instruments OMAP 3503 ARM Cortex-A8 CPU with 256MB RAM. It supports both WiFi and Fast Ethernet and runs Linux kernel 3.0. In our tests, we used CCNx version 0.4.0. Our code is portable and has been also successfully deployed on both Intel-based Mac and Linux computers.

Interests are signed using RSA with a 1024-bit modulus and a public exponent of 3. This allows for efficient verification, requiring only requires two multiplications. The hash algorithm used in signature computation is SHA-256. We implemented symmetric authenticated interests using HMAC with SHA-256.

Each command is associated with a state variable, which contains current time in seconds/microseconds, sequence number corresponding to the command and an estimated round-trip time between application and light fixture expressed in milliseconds.

Commands are treated as opaque binary strings by *NameCrypt*. Since CCNx allows name components to be binary blobs, commands and authenticators are not encoded in any printable format.

NameCrypt also implements efficient authentication for fixture acks. Encrypted challenges are implemented using AES-128 as block cipher and SHA-256 as the hash function. Hash-chain-based authenticated acknowledgments are based on SHA-256. Our implementation uses a very simple “pebbling” technique to reduce the cost of acknowledgment generation: rather than storing the whole chain (i.e., performing a full lookup) or computing pre-image $H^i(x)$ from x , fixtures store every 100-th link in the chain. This way, returning $H^i(x)$ requires, on average, computing 50 hashes. We emphasize that there are more efficient pebbling technique (e.g., [21]) that we can adopt without any changes to our protocol.

4.2 Performance Evaluation

Experiments were performed on a commodity laptop, which runs the sequencer, controller and fader (see Section 4), and on a low-powered embedded system (representative of a lighting fixture or a low-power fixture controller). The laptop uses a 2.53GHz Intel Core2Duo CPU. The embedded system uses an the Overo platform detailed above. We evaluated command and ack authentication micro-benchmarks and discuss performance considerations.

Table 3 shows the results of micro-benchmarks in command authentication. Time required to generate an RSA signature on the Intel platform is comparable to typical network latency in a LAN and does not significantly affect the performance of the whole protocol. Verification is well below typical network latency on both platforms. For this reason, we believe that features provided by digital signatures and their relative low cost justifies their use in an environment where commands are generated on non-constrained device. On the other hand, benchmarks show that low-power devices are not well-suited for generating real-time signatures on commands. In this case, we recommend the use of HMAC.

Symmetric authentication incurs negligible performance impact. Fixtures must generate a symmetric key for each application starting from their secret. This requires far less than a millisecond on our test devices.

Table 4 shows timing results of ack authentication mechanisms. Similar to command authentication, digital signatures do not introduce any significant delay on the Intel platform, while signature generation is relatively expensive on the ARM. Whenever viable, our tests show HMAC provides adequate performance. However when public verifiability is required and standard signatures are too expensive, encrypted challenges are an appealing option, as shown in Table 4.

Applications and fixtures may want to rely on hash chains for added functionality. In this case, benchmarks clearly show that fixtures should use an efficient representation of the hash chain, i.e., one that does not force the fixture to re-generate a large portion of the chain from the secret seed x . With the “pebbling” technique, challenge-response requires less than a millisecond. All other operations incur very low overhead on both platforms.

5 Security Analysis

Trust model. In our model the relationship between content objects can be represented in a undirected graph where each content object is a vertex. Various vertices are connected through public-key signatures. In particular, each edge represents a pair (public-key signature, key locator) – i.e., a signature is an edge from a content object carrying a public key to a content object signed by that key, while a key locator is an edge from a content object to the content object carrying the corresponding verification key. Vertices carrying public key can have multiple edges, while vertices representing regular content have only one edge.

Operation	Intel Core2Duo	ARM Cortex A8
Sign content object (RSA 1024)	2.018 ms	26.418 ms
Verify content object (RSA 1024)	0.046 ms	1.301 ms
Authenticate/verify (HMAC)	0.006 ms	0.070 ms
Encrypted challenge – create	0.003 ms	0.043 ms
Encrypted challenge – answer	0.001 ms	0.015 ms
Encrypted challenge – verify	0.001 ms	0.015 ms
Hash chain – create	11.350 ms	88.407 ms
Hash chain – answer w/ lookup	<0.001 ms	<0.001 ms
Hash chain – answer w/o full lookup	5.104 ms	44.196 ms
Hash chain – answer w/ partitioning	0.052 ms	0.443 ms
Hash chain – verify	0.001 ms	0.010 ms

Table 4: Performance analysis of ack authentication. RSA with public exponent 3; hash chains with 10,000 elements

A proof of ownership of $name_P$ from producer P to a challenger C consists of a graph with a path from a vertex V_i , which represents the content object named by C under $name_P$, to vertex V_0 , which denotes the content object containing TTP’s key. The prefix of the NDN name of each vertex along the path from V_0 to V_i must be the full namespace of the previous vertex, with the exception of the vertex signed by the TTP. For this reason, only the TTP or the owner of a namespace $name$ can elect a user to be the owner of namespace $name/child$.

Key Attribute and Access control policies. Let application App be the owner of a namespace $name_{App}$, which represents a set of pairs attribute/value as defined in Section 3.5. The goal of our key attribute and access control policy mechanism is to guarantee that – without the ownership of additional namespaces – App can only assign new namespaces to other owners as long as such namespaces identify more restrictive attribute/value pairs (as defined by specific applications) than $name_{App}$.

The security of our key attribute and access control policy mechanism is based on the security of our trust model. In particular, our trust model guarantees that App cannot become owner of a namespace with a prefix that differs from the full name of $name_{App}$ without receiving any additional namespace ownership from TTP or other applications. This is implied by the well-formedness of the proof of ownership: each vertex on the path from the TTP’s key to App’s key must represent a namespace with a prefix corresponding to the full namespace of the previous vertex – with the exception of the first vertex after TTP’s key. A proof that shows App’s ownership of a namespace with a prefix that does not correspond to the full name of the parent namespace would clearly be invalid.

Symmetric authentication mechanism. We now analyze the security of our symmetric interest authentication mechanism as defined in Section 3.6. Consider a malicious application \mathcal{ADV} whose goal is to issue correctly authenticated commands for Fix in a namespace for which \mathcal{ADV} has never received the corresponding symmetric key from Fix. We model this adversary as follows: \mathcal{ADV} is allowed to query Fix with any arbitrary namespace $name_i$ and receive the corresponding key $k_i = \text{PRF}_{k_{\text{Fix}}}(name_i)$. Eventually \mathcal{ADV} selects a namespace n_{adv} , never queried to Fix, under which it wants to be challenged. After revealing n_{adv} , \mathcal{ADV} is allowed access to oracle $O^{(n_{adv})}(cmd_i)$, which outputs $m_i = \text{MAC}_{k_{adv}}(cmd_i)$ where $k_{adv} = \text{PRF}_{k_{\text{Fix}}}(n_{adv})$. n_{adv} cannot be included in subsequent queries to Fix. The goal of \mathcal{ADV} is to issue a pair (cmd_{adv}, m_{adv}) with $m_{adv} = \text{MAC}_{k_{adv}}(cmd_{adv})$, with cmd never queried before to $O^{(n_{adv})}$. In other words, \mathcal{ADV} can obtain the symmetric key corresponding to any namespace of its choice from Fix. Also, \mathcal{ADV}

can choose a “target namespace”, n_{adv} , and request commands to be authenticated using $O^{(n_{adv})}$. \mathcal{ADV} ’s goal is to issue a *never requested* authenticated command in namespace n_{adv} .

We now sketch how to construct a simulator S that interacts with challenger C for a secure MAC with key k and uses \mathcal{ADV} as a subroutine. For each query n_i from \mathcal{ADV} to Fix where n_i was never asked before, S responds with a random value k_i . S stores pairs (n_i, k_i) in a table that is used to respond consistently to subsequent queries. Assuming that the PRF used by Fix is secure, \mathcal{ADV} cannot distinguish between (truly) random responses from S and the pseudorandom responses from Fix. S implements $O^{(n_{adv})}(cmd_i)$ by querying challenger C on cmd_i . C returns $m = \text{MAC}_k(cmd_i)$ and S forwards it to \mathcal{ADV} . This way, S implicitly sets $\text{PRF}_{k_{\text{Fix}}}(n_{adv}) = k$ without knowing k . Eventually \mathcal{ADV} outputs (cmd_{adv}, m_{adv}) , and S outputs (cmd_{adv}, m_{adv}) as its response to C . It is easy to see that (cmd_{adv}, m_{adv}) is a well formed pair command/authenticator iff $\text{MAC}_k(cmd_{adv}) = m_{adv}$. Since we assume that MAC is secure, \mathcal{ADV} can only output such pair with negligible probability.

Encryption-based Ack Authentication. The goal of our encryption-based ack authentication protocol is to prevent an adversary \mathcal{ADV} from acknowledging a command cmd on behalf of – and without any help from – Fix. We argue that, if H is hard to invert and E is a secure block cipher – i.e., E implements a pseudorandom permutation – then \mathcal{ADV} has only negligible probability of generating a valid ack for cmd .³

We model \mathcal{ADV} as follows: it interacts with a challenger by requesting pairs $(y_i, z_i) = (E_k(x_i), H(x_i))$ for a value x_i of its choice. Eventually, \mathcal{ADV} receives a challenge (y, z) corresponding to a random x selected by the challenger. \mathcal{ADV} ’s goal is to output a value x' such that $H(x') = z$.

We argue that the existence of such adversary – which outputs a correct z' with non-negligible probability – would either violate the security of the block cipher or the one-wayness of the hash function. In particular, it would allow the construction of either a distinguisher that sets apart the output of E from a truly random string of the same size, or a simulator S_{adv} that inverts H.

First, we observe that \mathcal{ADV} cannot distinguish between pairs $(E_k(x), H(x))$ and $(r, H(x))$ where r is a random value of the same size as the block size of E. If \mathcal{ADV} could distinguish between the two with non-negligible advantage, it could be trivially used to build a distinguisher that sets apart the output of a pseudorandom permutation from a truly random string of the same size.

We define S_{adv} as follows: it receives a challenge $y = H(x)$ for a random x chosen by a challenger C . S_{adv} must compute a value x' such that $H(x') = y$. S_{adv} answers \mathcal{ADV} ’s queries x_i by returning $(r_i, H(x_i))$, where r_i is a random value of appropriate size. It also stores pairs (x_i, r_i) to answer consistently to subsequent queries. \mathcal{ADV} can only detect the simulation by determining that r_i is not the output of E. However, we argued above that this is possible only with negligible probability.

Eventually \mathcal{ADV} requests a challenge and S_{adv} responds with (y, r) for a fresh random r . \mathcal{ADV} can detect the simulation only if there exist an index i such that $x_i = x$, which can happen only with negligible probability.

Finally, \mathcal{ADV} outputs x' . S_{adv} outputs the same value to answer the challenge from C . It is easy to see that S_{adv} successfully inverts $H(\cdot)$ iff \mathcal{ADV} ’s output is correct. Therefore, \mathcal{ADV} can only forge authenticated acks with negligible probability.

³ Our scheme is also secure if E is any CPA-secure encryption scheme with efficiently sampleable ciphertext space. In this case, the value r in pairs $(r, H(x))$ in the security discussion must be selected from the ciphertext space of E.

Hash-chain-based Ack Authentication. We consider an adversary \mathcal{ADV} that can control the communication channel between App and Fix – e.g. it can drop, modify, inject, delay, or replay any packet between the two parties. The goal of \mathcal{ADV} is to produce an authenticated ack to any command issued by App for which Fix has not yet responded.

In our hash-chain-based protocol, upon completion of an authenticated command \mathcal{ADV} from App containing challenge $H^i(x)$, Fix reveals preimage p in its acknowledgment, such that $H(p) = H^i(x)$, i.e., $p = H^{i-1}(x)$. If App fails to receive an authenticated ack from Fix, it reissues cmd until either: 1. App receives p ; or 2. after a predefined timeout. We emphasize that App does not issue a new command cmd until it receives an ack for cmd from Fix.

If App deviates from this behavior, the protocol is insecure: consider the scenario where App issues cmd for which it receives no ack. Then App issues a new command $cmd' \neq c$. Since no preimage was received for cmd , App has no other option but to construct cmd' using the same challenge as in cmd . Assume that Fix responded to cmd and \mathcal{ADV} dropped the ack after learning p . From now on \mathcal{ADV} can acknowledge any command issued with the same challenge, which is clearly undesirable.

Since the same challenge is never used twice, \mathcal{ADV} cannot issue an authenticated ack without receiving the corresponding preimage from Fix. However, since commands are authenticated, Fix reveals a preimage only if the command has not been altered by \mathcal{ADV} and has been successfully executed. For this reason, \mathcal{ADV} can only acknowledge commands successfully executed by Fix.

We now show App is able to prove to a third party its successful interaction with Fix. In order to do so, App must produce the preimage $p = H^{i-1}(x)$, and an anchor $H^\ell(x)$ signed by AM. We point out that App is unable to produce p unless it receives an authenticated ack from Fix – which contains p .

Assume that, given $H^i(x)$, App can produce p without interacting with Fix. Since App has no additional information on the hash chain, the only way for App to compute p from $H^i(x)$ is to invert $H(\cdot)$. Therefore App can only succeed in producing a valid proof of interaction with negligible probability.

We emphasize that p does not depend on the actual command it acknowledges. For this reason, a proof of interaction does not suffice to determine which command it corresponds to. However if A is authorized to issue only a command cmd with no parameters, then p successfully shows that Fix must have acknowledged cmd .

6 Related Work

As discussed in Section 1.3, we target a hybrid design space that draws from both the lighting control and building automation worlds. Here we summarize related work in both areas.

Digital lighting control plays a key role in both building automation and entertainment settings. (In fact, their use in the latter often leads the way in technical innovation [25, 22].) Most current protocols are descended from legacy control architectures based on serial communication [25]. These legacy architectures rely on a separate communication infrastructure. In this context, availability, integrity, privacy and authentication are not an issue since outsiders are assumed to have no access to the communication media.

In order to reduce deployment cost and to evolve such architectures to building- and campus-wide installations, vendors have introduced ways to transport legacy protocol data over IP [2]. This provides great flexibility, allowing devices to reuse already-deployed communication infrastructures such as Ethernet, WiFi, or other IP-compatible media. In this transition from serial to IP, vendors have rarely implemented additional security measures [32], likely to avoid

increasing development and deployment costs and in order to maximize compatibility. As a consequence, such protocols must be often run over VLANs, VPNs [20], IPsec [32] or physically segregated networks. Physical segregation is often difficult or even impossible when protocols are running over RF. In this case several of them have been shown to be insecure [32].

Below we introduce the main protocols in use and briefly discuss their security features or lack thereof.

DMX/DMX512. Most modern lighting control protocols descend from or implement DMX512 [9], which is an industry standard multidrop serial protocol based on RS-485. In the past, each serial DMX cable provided 512 one-byte control channels updated at a maximum rate of 44Hz. In DMX nomenclature a link addressing 512 devices is considered a *universe*. Modern lighting control systems encapsulate DMX payload over modern media such as wired/wireless Ethernet [2] or RF [10]. This has resulted in various competing technologies such as Art-Net [2], ACN [25], ETCNet/ETCNet2 [12] which bridge lighting systems and allow them to coexist with newer technology and integrate into BAS.

Art-Net. Art-Net [2] is a proprietary protocol with open specifications for transporting DMX signal over UDP. It uses wireless/wired Ethernet as communication medium; devices self-configure IP addresses based on the hardware MAC address, and use broadcast transmission for communication. Art-Net does not offer any form of authentication or encryption. Reliance on broadcast and lack of security clearly shows that Art-Net was designed to run on local (access restricted) networks. In [31], Newton shows how to run Art-Net over VPN to overcome some of the protocol's limitations.

Architecture for Control Networks (ACN). ACN [25] is a set of ANSI standards which define a protocol suite for controlling lighting, networked entertainment devices, and existing control systems. It has been designed to address several shortcomings of existing lighting control protocols, specifically (1) having both an open protocol and specification, (2) media agnostic control, and (3) generalizing to any device that can be controlled [25]. ACN defines several protocols on top UDP, and therefore naturally extends to any medium that can carry IP communication. There are several implementations of ACN. As an example, OpenACN [33] and the current generation of ETCNet [12] implement the ACN standard.

Security is not addressed in this standard, which assumes that ACN data is transported over a secure network.

Proprietary Protocols. Alongside aforementioned lighting control protocols, there are proprietary vendor-centric solutions, such as Philips Dynalite and Philips KiNET. Philips Dynalite is designed for controlling lighting, interfacing to HVAC, security, fire detection systems, and other building sensors [11]. The control protocol uses multi-drop serial over twisted pair and its characteristics are similar to DMX. Philips KiNET [26] is a contemporary proprietary lighting protocol based on the Philips Color Kinetics platform for LED lighting technology. It uses Ethernet as its communication media. To the best of our knowledge, KiNET does not offer any form of authentication or encryption between devices.

6.1 Common BAS protocols

In contrast with lighting, BAS protocols interconnect multitudes of sensors and actuators across a building, including HVAC, building controls, as well as home and office lighting. The turnkey nature of these systems combined with the need to inter-operate in more complex buildings has led to both manufacturer-specific protocols and standardization of Internet protocols in order to converge and distribute control to these systems. As a result, there

are several BAS that have gained widespread adoption. We review most prominent solutions: KNX [27] (formerly EIB), Fieldbus [13], LonTalk/LonWorks [29], and BACNet [1].

BACNet. BACNet [1] is an open standard specifying a protocol at the backbone level to communicate with devices at the control level. Although fixtures can support BACNet directly, one of the primary goals of BACNet is interoperability with other BAS. For the backbone level, various data link and physical media technologies have been specified based on existing technology, such as Ethernet, IP (separately as BACNet/IP), LonTalk, etc.

This standard supports encryption and authentication, although it has been shown to be insecure [32, 20, 39, 15, 17]. DES is the only supported block cipher, and the authentication technique is susceptible to man-in-the-middle attacks [32]. Moreover, the protocol does not offer protection against interleaving and replay attacks [32]. In the last decade, there have been several efforts aimed at securing BACNet, e.g. Robin et al. [34].

KNX/EIB. KNX began as an open standard – superseding the legacy European Installation Bus (EIB) – converging several existing standards in home automation and intelligent buildings. Typical KNX deployments are used to manage a multitude of building control applications, such as lighting, HVAC, energy management and metering. KNX supports a variety of communication media such as Radio (KNX-RF), Infrared and Powerline.

As outlined in [16, 15, 17], KNX provides no data security. The control communication to the fixture is limited to a rudimentary access control scheme. The access control scheme uses access levels to define privileges ranging from 0 (highest privilege) to 255 (lowest privilege). Each access level supports a 4-byte password stored and transmitted in clear-text. Two extensions proposed to improve the security of KNX are EIBsec [16] and a method combining Diffie-Hellman and AES [4].

Cavalieri et al. [4] present an extension to the KNX application layer that uses a configuration manager placed in the control plane to distribute keys to fixtures using Diffie-Hellman [7]. This key exchange is used to establish a secure challenge, followed by a challenge-response protocol to authenticate the parties. A long-term key is established for use with AES. This proposal specifies no security model, and protocol details are insufficient to evaluate its security.

LonTalk/LonWorks. LonTalk is the communication protocol for the LonWorks BAS. It supports several media types, such as RF, Infrared, Coaxial cable, and Fiber Optics. It is a major competitor of KNX and is currently in widespread use in building automation.

As specified in [32, 35, 16, 15, 17], LonTalk provides minimal security. The only security feature is a protocol for data origin authentication in both unicast and multicast. Each entity is limited to a single key of up to 48 bits. All entities must share the same key if they want to verify messages amongst each other. No security mechanism is provided to distribute keys. Thus, each device must be bootstrapped off-line, in a secure environment. Significant overhead is incurred for authentication as the protocol requires a 4-round challenge-response protocol invoked for each message the sender transmits.

Fieldbus. Fieldbus [13] is traditionally used in industrial automation and control. It is an open architecture with published IEC standards, which has led to industry adoption and proliferation of several vendors. Its specification defines a communication bus protocol for monitoring sensors and operating actuators. The technology has been later adapted to building automation and control.

As outlined by Tretyl et al. [37], in-band security mechanisms for Fieldbus implementations offer weak security. Most implementations offer rudimentary access control and authentication where passwords are sent over the network in cleartext. Security of a typical Fieldbus deployment relies on network isolation. [37] also proposes the use of standard mechanisms for securing communications in Fieldbus deployments, e.g., SSL/TLS and IPSec.

Proprietary Protocols. There are several contenders in the BAS ecosystem with proprietary protocols, such as Siemens and Honeywell. Since published specifications are scarce we do not overview these systems.

7 Summary and Future Work

This paper focused on securing instrumented environments connected via Content-Centric Networking (CCN), motivated by the increasing integration of Building Automation Systems (BAS) with enterprise networks and the Internet. In particular, we explore lighting systems over Named-Data Networking (NDN), a prominent instance of CCN. We identified security requirements in lighting control and constructed a concrete NDN-based security architecture. We then analyzed its security properties and reported on the prototype implementation and experimental results.

Clearly, this work represents only the initial step towards assessing suitability of NDN for communication settings far from its *forte* of content distribution. Much more work is needed to securely adapt NDN to other types of instrumented environments. Lighting control is, in some ways, simpler than other BAS types. For example, we assumed a limited model of feedback in which most of the time command acks are basically one-bit values. This allowed us to use tricks based on hash chains or *encrypted challenges* to obtain efficiency. In other circumstances, acks may be more expressive.

Our current design does not support multicast communication. In order for a group of fixtures to be synchronously commanded by the same application, each fixture needs to issue a separate interest to the application at roughly the same time. The application could then issue a command that would reach all “interested” fixtures.

Another direction worth exploring is the utility of long-lived interests, i.e., interests that establish state in NDN routers but do not expire quickly. This can be useful if we allow fixtures to issue interests to controlling applications (rather than the other way around as we do now). A fixture Fix would issue an interest to App and the latter would only emit corresponding content when it has a command for Fix. This would require Fix to periodically refresh interests as they expire and get flushed by routers.

While not discussed in detail here, NDN offers other significant benefits to BAS applications that have motivated this exploration, such as providing network access to sensing and actuation points via application-assigned data names, without the need to specify IP host addresses and port numbers for gateways.

8 References

- [1] ANSI. Standard 135-1995, BACnet a data communication protocol for building automation and control networks, 1995.
- [2] Specification for the Art-Net 3 Ethernet Communication Standard. <http://www.artisticlicence.com>. Retrieved Feb. 2012.
- [3] N. Asokan, G. Tsudik, and M. Waidner. Server-supported signatures. *Journal of Computer Security*, 5:131–143, 1996.
- [4] S. Cavalieri and G. Cutuli. Implementing encryption and authentication in KNX using Diffie-Hellman and AES algorithms. In *IEEE IECON*, pages 2459–2464, Nov. 2009.
- [5] Y. Challal, A. Bouabdallah, and Y. Hinard. Efficient multicast source authentication using layered hash-chaining scheme. In *IEEE LCN*, Nov. 2004.
- [6] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun. ANDaNA: Anonymous named data networking application. In *NDSS*, 2012.
- [7] W. Diffie and M. Hellman. New directions in cryptography, 1976.
- [8] X. Ding, D. Mazzocchi, and G. Tsudik. Experimenting with server-aided signatures. In *NDSS*, 2002.
- [9] DMX512-A. <http://www.opendmx.net/index.php/DMX512-A>. Retrieved Feb. 2012.
- [10] Lighting systems made easy a guide to lighting installations. <http://www.leprecon.com/catalogs/280075BLightingMadeEasy.pdf>. Retrieved Feb. 2012.
- [11] An introduction to the Philips Dynalite control system. http://www.lighting.philips.com/pwc_li/main/subsites/dynalite/library_support/assets/general_brochures/intro_to_phd_controls_systems.pdf. Retrieved Feb. 2012.
- [12] Electronic Theater Controls (ETCNet). <http://www.etcconnect.com/>. Retrieved Feb. 2012.
- [13] Fieldbus history. <http://www.fieldbus.org/>. Retrieved Feb. 2012.
- [14] C. Gentry and A. Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.
- [15] W. Granzer and W. Kastner. Security analysis of open building automation systems. In *SAFECOMP*, pages 303–316, 2010.
- [16] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Prais. Security in networked building automation systems. In *IEEE WFCS*, pages 283–292, Jun. 2006.
- [17] W. Granzer, F. Prais, and W. Kastner. Security in building automation systems. *IEEE IES*, 57(11):3622–3630, Nov. 2010.
- [18] M. Gritter and D. Cheriton. An architecture for content routing support in the internet. In *USENIX USITS*, 2001.
- [19] Gumstix Overo Air. http://www.gumstix.com/store/product_info.php?products_id=226. Retrieved Feb. 2012.
- [20] D. Holmberg and D. Evans. Bacnet wide area network security threat assessment. <http://fire.nist.gov/bfrlpubs/build03/art034.html>, Jul. 2003.
- [21] Y. Hu, M. Jakobsson, and A. Perrig. Efficient constructions for one-way hash chains. In *ACNS*, pages 423–441, 2005.
- [22] J. Huntington. *Control Systems for Live Entertainment*, 3rd Edition. Burlington, MA: Focal Press, 2007.
- [23] V. Jacobson, D. Smetters, N. Briggs, M. Plass, J. Thornton, and R. Braynard. VoCCN: Voice-over content centric networks. In *ReArch*, 2009.
- [24] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking named content. In *ACM CoNEXT*, 2009.
- [25] W. Jiang, Y. Jiang, and H. Ren. Analysis and prospect of control system for stage lighting. In *IEEE CISP*, volume 8, pages 3923–3929, Oct. 2010.
- [26] Philips multi-protocol converter. http://www.colorkinetics.com/support/datasheets/Multi-Protocol_Converter_SpecSheet.pdf. Retrieved Feb. 2012.
- [27] KNX system specifications v 3.0. <http://www.knx.org/downloads-support/downloads/>.
- [28] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM*, volume 37, pages 181–192. ACM, 2007.
- [29] iLON 600 LonWorks/IP server models 760x. <http://www.echelon.com/support/documentation/datashts/7260x.pdf>. Retrieved Feb. 2012.
- [30] Named data networking project (NDN). <http://named-data.org>. Retrieved Feb. 2012.
- [31] S. Newton. Art-net and wireless routers. In *IEEE APCC*, pages 857–861, Oct. 2005.
- [32] N. Okabe, S. Sakane, K. Miyazawa, K. Kamada, A. Inoue, and M. Ishiyama. Security architecture for control networks using IPsec and KINK. In *IEEE SAINT*, pages 414–420, 2005.
- [33] Open ACN project. <http://openacn.engarts.com>. Retrieved Feb. 2012.
- [34] D. Robin. Internet security protocols for BACnet, BACnet committee SSPC 135 document number DR-028-1, 2002.
- [35] C. Schwaiger and A. Treytl. Smart card based security for fieldbus systems. In *IEEE ETFA*, volume 1, pages 398–406, Sept. 2003.
- [36] D. Smetters. CCNx access control specifications. Technical report, PARC, 2010.
- [37] A. Treytl, T. Sauter, and C. Schwaiger. Security measures for industrial fieldbus systems - state of the art and solutions for IP-based approaches. In *IEEE WFCS*, pages 201–209, Sept. 2004.
- [38] A. Yavuz and P. Ning. Hash-based sequential aggregate and forward secure signature for unattended wireless sensor networks. In *ICST Mobiquitous*, pages 1–10, Jul. 2009.
- [39] J. Zachary, R. Brooks, and D. Thompson. Secure integration of building network into the global internet. <http://fire.nist.gov/bfrlpubs/build03/art027.html>, Oct. 2002.